SENG479 – GAME PROGRAMMING, 2025-2026 FALL TERM

FINAL PROJECT

PROJECT 2: ROGUELIKE DUNGEON CRAWLER

Overview

In this project, you will create a roguelike dungeon crawler where players explore procedurally connected rooms, battle enemies, collect items, and attempt to survive as long as possible. The game follows the rogue-like tradition of permadeath, meaning when the player dies, they must start over from the beginning. A single run should take approximately five to ten minutes to complete.

Learning Objectives

Upon completing this project, you will be able to:

1. Implement procedural level generation using room templates and random connections
2. Design enemy artificial intelligence using state machines
3. Create satisfying combat mechanics with appropriate visual and audio feedback
4. Manage game state including player statistics, inventory, and progression
5. Implement tile-based rendering and collision systems
6. Apply game feel techniques such as screen shake, hit pause, and knockback

Technical Requirements

Your game must be developed using Raylib with either C or C++. You may use additional single-header libraries only with prior instructor approval. The game must maintain a minimum of 60 frames per second during normal gameplay.

Required Features

Level Generation

Your dungeon must consist of connected rooms. You may use fully procedural generation or connect pre-designed room templates in random configurations. Each run must feel different from the previous run through randomized room layouts or room selection. The player must be able to navigate between rooms through doorways or transitions. Each run must include a minimum of eight rooms before the player can reach the exit or final boss. The minimap or some indication of explored rooms is recommended but not required.

Player Character

The player must have visible health that can be damaged by enemies. The player must have at least one attack method, either melee or ranged. Player movement must feel responsive with appropriate acceleration and deceleration. The player must have a brief invincibility period after taking damage to prevent instant death from overlapping enemies. The player's current health must always be visible on screen.

Enemy System

You must implement at least three distinct enemy types. Each enemy type must have different behavior, such as one that charges directly at the player, one that maintains distance and attacks from range, and one that moves in patterns. Enemies must use a state machine architecture with at least two states such as idle, chasing, attacking, or fleeing. Enemies must provide clear visual indication before attacking to give the player a chance to react. Defeated enemies must have a chance to drop items.

Item System

The game must include collectible items that affect gameplay. At minimum, you must implement health restoration items and at least one type of item that improves player combat ability such as increased damage or attack speed. Items must have a visible appearance in the world before collection. When the player collects an item, there must be visual and audio feedback.

Equipment or Inventory

The player must be able to hold or equip items that persist between rooms. This can be a simple weapon swap system, an inventory screen, or a passive upgrade system. The current equipment or active items must be visible in the user interface.

Game Flow

The game must include a main menu with options to start a new run and quit. When the player dies, a game over screen must display statistics such as rooms explored, enemies defeated, or time survived. The game over screen must offer the option to start a new run or return to the main menu. If the player reaches the end goal, a victory screen must appear with appropriate celebration.

Code Organization

Your project must be organized across multiple source files. You should have separate files for player logic, enemy management including AI states, level generation, item and inventory systems, and rendering. State machine code should be cleanly separated from other logic. All functions must have clear, descriptive names.

Submission Requirements

You must submit your project as a **GitHub repository**. Create a new repository using the format **GroupID-LastName-FirstName-ProjectName** (for example, 1-Smith-John-TowerDefense). The repository must be set to private, and you must add the instructor as a collaborator. (**My Github Profile: https://github.com/batuhanhangun**)

Your repository must contain all source code files organized in a logical folder structure, all asset files including images and sounds in an assets folder, **a README.md file** at the root with build instructions, gameplay explanation, controls, and credits for any external assets, and **a ".gitignore"** file appropriate for C or C++ projects.

Your commit history is part of your submission. You are expected to make regular commits throughout the development process that show incremental progress. Repositories with only one or two commits containing the entire project will be flagged for review, as this does not demonstrate a healthy development process.

Your final commit must include a compiled executable in a "build" or "bin" folder, or your README must include exact compilation instructions that work on lab machines.

Submit the link to your repository through the course learning management system (MS Teams Assignment) before the deadline.

Game Design Document

As part of your submission, you must create a Game Design Document (GDD) that describes your game's design, mechanics, and creative vision. The GDD serves as both a planning tool during development and a record of your design decisions for evaluation. Professional game developers create GDDs before and during production to communicate their vision to team members and stakeholders, and learning to articulate your design in written form is an essential skill.

You are free to choose any format for your GDD. You may use a traditional structured document, a wiki-style page, a visual diagram-heavy approach, or any other format that effectively communicates your game's design. The format you choose should match the complexity of your game and your personal communication style. Some students prefer formal documents with numbered sections, while others prefer more visual or casual formats. All approaches are acceptable if the required content is present and clearly communicated.

Your GDD must be saved as a Markdown file named GDD.md and placed in the root directory of your GitHub repository alongside your README.md file. Using Markdown allows your document to be rendered nicely when viewed on GitHub and keeps all project documentation in a consistent format.

Your GDD must address the following content areas, though you may organize and present them however you see fit.

Game Overview: Provide a brief description of your game including its genre, theme, and core concept. A reader should understand what kind of game you make and what makes it interesting after reading this section.

Gameplay Mechanics: Describe how your game plays. Explain the core mechanics, what the player can do, and how different systems interact with each other. This section should give a clear picture of the moment-to-moment gameplay experience.

Game Elements: Document the specific elements in your game such as enemy types, tower types, items, levels, arenas, resources, or whatever elements are relevant to your chosen project. For each element, briefly describe its purpose and how it contributes to the gameplay.

Progression and Goals: Explain how the player progresses through your game and what they are trying to achieve. Describe win conditions, lose conditions, difficulty progression, or scoring systems as applicable to your project.

Controls: List of all player controls and input methods. For multiplayer projects, document the control schemes for each player.

Visual and Audio Style: Describe the aesthetic choices you made for your game. Explain the visual style, color palette choices, and how audio contributes to the experience. You do not need to elaborate concept art, but you should articulate the creative direction you pursued.

Design Challenges and Solutions: Reflect on at least one significant design challenge you encountered during development and explain how you resolved it. This could involve balancing issues, mechanics that did not work as planned, or creative problems you had to solve. This section demonstrates your design thinking process.

The GDD will be evaluated as part of your Documentation grade. A complete GDD that clearly communicates all required content areas will receive full marks. The quality of your writing, clarity of explanation, and evidence of thoughtful design decisions will be considered. Superficial or incomplete GDDs will receive reduced marks.

Your GDD should be written as you develop the game, not as an afterthought the night before submission. The document should reflect genuine planning and design work. GDDs that appear to be reverse engineered from finished code rather than used as actual design documents will be viewed skeptically.

Academic Integrity

All code must be your own original work. You may reference Raylib documentation and examples. You may discuss general concepts with classmates but may not share code. Any external code snippets must be cited in comments. Violations will result in a zero grade and referral to academic affairs.